Unifying type systems for mobile processes

Emmanuel Beffara

Unifying type systems for mobile processes

Emmanuel Beffara

April 20, 2015

Abstract. We present a unifying framework for type systems for process calculi. The core of the system provides an accurate correspondence between essentially functional processes and linear logic proofs; fragments of this system correspond to previously known connections between proofs and processes. We show how the addition of extra logical axioms can widen the class of typeable processes in exchange for the loss of some computational properties like lock-freeness or termination, allowing us to see various well studied systems (like i/o types, linearity, control) as instances of a general pattern. This suggests unified methods for extending existing type systems with new features while staying in a well structured environment and constitutes a step towards the study of denotational semantics of processes using proof-theoretical methods.

1 Introduction

Process calculi are a wide range of formalisms designed to model concurrent systems and reason about them by means of term rewriting. Their applications are diverse, from the semantics of proof systems to the conception of concrete programming languages. Type systems for such calculi are therefore a wide domain, with systems of different kinds designed to capture different behaviours and ensure different properties of processes: basic interfacing, input-output discipline [25], linearity [22], lock-freeness [21], termination [10], respect of communication protocols [17, 20], functional or sequential behaviour [5, 19, 28].

In order to better understand the diversity of calculi and uncover basic structures and general patterns, many authors have searched for languages with simpler or more general theory in which the most features could be expressed by means of restrictions or codings: asynchrony [7], internal mobility [26], name fusions [13, 24], solos [23], etc. It is natural to search for similar unification in the realm of type systems, and the aim of this paper is to make a step towards this long-term objective. Our ideal system would be simple enough so that general properties could be reasonably easy to obtain and expressive enough so that most interesting type systems could naturally be expressed in it in a structured way.

For this purpose, we will take inspiration and tools in proof theory. A useful analogue is the famous and fruitful Curry-Howard correspondence: at the core is the simply typed λ-calculus, which matches minimal intuitionistic logic and ensures strong normalisation. The type language can be extended for expressiveness (with quantifiers, dependent types, polymorphism, etc.), classical logic can be embedded in it by CPS translation or by adding logical rules. Furthermore, extending it with a simple type equation $D = D \to D$ yields the full untyped calculus where normalisation is lost, but the identification of this equation leads to the definition of abstract structures that are useful for denotational semantics.

We claim that the analogue of simple types for process calculi is to be found in linear logic [14], and we propose a new implementation of this idea. Of course, term assignment systems for linear logic proofs have been proposed in the past by various authors [4, 2, 8] but no such system

has yet appeared as a satisfactory type system for processes (because of too much constraint on the syntactic structure of terms), with the notable exception of Honda and Laurent's result [18] which precisely matches a proper type system for the π-calculus with a meaningful class of proof nets.

The novelty of our approach is to distinguish two aspects in typing: firstly we have a typing rule for each syntactic construct independently, secondly we have a subtyping relation that implements logical reasoning without affecting the structure of terms; this subtyping is nothing else than entailment in linear logic (actually a reasonable fragment of it), which allows to use all existing theory for reasoning about it. In this method, we insist on treating seriously the fundamental structures of both the process calculus and the logic, the fundamental example being that typing is preserved both by structural congruence on processes and by logical isomorphism between types (and these are closely related). This is necessary for developing logic-based semantics of processes in future works, using existing tools and methods from the semantics of proofs and of processes.

This paper is divided in two parts. In section 2 we define our basic type system for the polyadic π-calculus and we discuss variations around the same principles for alternative calculi. In section 3 we review several type systems and term assignment systems and show how they fit in our framework, by means of extra logical axioms and syntactic restrictions. Section 4 concludes by discussing shortcomings, extensions and ideas for future work.

2 The basic typed calculus

2.1 Syntax

Processes are terms of the standard polyadic π-calculus with input-guarded replication (and no sum in the present paper), with type annotations on name creation. In our type system, we will derive judgements of the form $E \vdash P$ where E is an environment type and P is a process term. Such a judgement is to be understood as "P is well-formed under the contract of the environment E". Environment types are made of capability assignments of the shape $x : T$ where x is a channel name and T a capability type, combined using logical connectives. Capability types consist of an input or output capability (written $\downarrow$ and $\uparrow$ respectively) together with a behaviour type for the data that is communicated, and behaviour types are capability types combined using logical connectives.

1 **Definition (typed terms).** The grammar of types and processes is defined in table 1.

Remark that we do *not* force each channel name to occur only once in an environment type, and this is a fundamental feature of our system. It notably allows name substitution $E[\vec{u}/\vec{x}]$ to make sense even when it equalises some names.

The name creation operator $(\boldsymbol{\nu} x)$ is annotated with a type A and a kind k that distinguishes between linear and non-linear channels. Contrary to usual practice, the type A is not the type that x itself will have, but the type of the data that x will transport. In statements where the kind and type of a channel are unimportant, we use the standard notation $(\boldsymbol{\nu} x)$.

The logical connectives used in environments and behaviours are those of multiplicative-exponential linear logic. This logic, recalled in table 2, is used to reason about behaviours of processes. The key ingredient is that logical consequence is interpreted as subtyping: if E and F are environments such that E entails F, then a process that respects F will respect E.

2 **Definition (subtyping).** The subtyping preorder $\leq$ over environments is such that $E \leq F$ holds when $\vdash E^{\perp}, F$ is provable in MELL using capability assignments as atomic formulas. The associated equivalence is written $\simeq$.

Capabilities:	$T, U := \downarrow A$	input
	$\uparrow A$	output
Behaviours:	$A, B := T,\ !T,\ ?T$	capability (linear, replicable, multiple)
	$A \otimes B,\ A \parr B$	concatenation (independent, correlated)
	$1,\ \bot$	empty tuple (neutral for $\otimes$ or $\parr$)
Environments:	$E, F := x : T,\ !x : T,\ ?x : T$	capability assignment
	$E \otimes F,\ E \parr F$	union (independent or correlated)
	$1,\ \bot$	empty environment (neutral for $\otimes$ or $\parr$)
Processes:	$P, Q := u(\vec{x}).P$	input prefix
	$!u(\vec{x}).P$	replicated input prefix
	$\bar{u}\langle\vec{v}\rangle.P$	output prefix
	0	inactive process
	$P \mid Q$	parallel composition
	$(\boldsymbol{\nu} x : A^k)P$	name creation, with $k \in \{1, \omega\}$

Table 1: The syntax of types and process terms

Formulas of MELL with capability assignments as atoms and where modalities ? and ! are only applied to literals (atoms and atom negations) will be called *environment formulas*, they will be useful for reasoning about typed processes. Environment types correspond to such formulas with only positive atoms, i.e. without negation.

3 **Definition (typing judgement).** Typing judgements have the shape $E \vdash P$ where E is an environment type and P is a process term. They are derived using the rules of table 3.

The notation $\vec{x} : A$ where A is a behaviour type stands for the environment type obtained by annotating each capability in A by a name in the sequence $\vec{x}$, respecting the left-to-right order, assuming that the length of $\vec{x}$ matches the number of capabilities in A. For instance, $xyz : (T \parr 1) \otimes ?U \otimes \bot \otimes V$ stands for $((x : T) \parr 1) \otimes ?(y : U) \otimes \bot \otimes (z : V)$.

Note that process terms have no type, in other words there is a unique type for processes which means "well-formed"; it is also the case for instance in i/o types with linearity, as studied in section 3.1. Of course, it would be strictly equivalent to consider that, in $E \vdash P$, the formula $E^\perp$ is the type of P: this is what usually happens in systems more oriented towards logic, like those studied in sections 3.2 and 3.3.

Remark that input and output capabilities are logically not dual, in the sense of being a negation of each other: $(u : \downarrow A)^\perp$ and $u : \uparrow A$ are just distinct literals. Actual duality between input and output is established by the typing rules for name creation, for instance NEW1 corresponds to setting the formula $x : \uparrow A \parr x : \downarrow A$ as an axiom, which does represent the creation of a name x with one occurrence of each capability, where capabilities are dual.

In the statements and proofs, the types for channels in premisses of NEW rules is used in many places. For readability and conciseness, we introduce the following notations:

$$[x]^1_A := x : \uparrow A \parr x : \downarrow A, \qquad [x]^\omega_A := !x : \uparrow A \parr ?x : \downarrow A.$$

In $[x]^k_A$, we may keep k or A implicit when the details are unimportant. This way, the rules NEW1 and NEWω are simplified into a single form:

$$\frac{[x]^k_A \otimes E \vdash P}{E \vdash (\boldsymbol{\nu} x : A^k)P}\ \text{NEW}k \qquad \text{where } x \text{ does not occur in } E.$$

Formulas, assuming a given set of atoms written $\alpha, \beta \ldots$:

$$A, B := \alpha,\ \ \alpha^\perp,\ \ A \otimes B,\ \ A \parr B,\ \ 1,\ \ \perp,\ \ !A,\ \ ?A$$

Linear negation $(\cdot)^\perp$ is the involution over formulas such that:

$$(\alpha^\perp)^\perp = \alpha \qquad (A \otimes B)^\perp = A^\perp \parr B^\perp \qquad 1^\perp = \perp \qquad (!A)^\perp = ?A^\perp$$

Sequents are finite multisets of formulas, they are proved using the following rules:

$$\frac{}{\vdash A^\perp, A}\ \text{AX} \qquad \frac{\vdash \Gamma, A \quad \vdash A^\perp, \Delta}{\vdash \Gamma, \Delta}\ \text{CUT} \qquad \frac{\vdash \Gamma, A \quad \vdash \Delta, B}{\vdash \Gamma, \Delta, A \otimes B}\ \otimes \qquad \frac{\vdash \Gamma, A, B}{\vdash \Gamma, A \parr B}\ \parr$$

$$\frac{}{\vdash 1}\ 1 \qquad \frac{\vdash \Gamma}{\vdash \Gamma, \perp}\ \perp \qquad \frac{\vdash \Gamma, A}{\vdash \Gamma, ?A}\ ? \qquad \frac{\vdash \Gamma}{\vdash \Gamma, ?A}\ \text{W} \qquad \frac{\vdash \Gamma, ?A, ?A}{\vdash \Gamma, ?A}\ \text{C} \qquad \frac{\vdash ?A_1, \ldots, ?A_n, B}{\vdash ?A_1, \ldots, ?A_n, !B}\ !$$

Table 2: Multiplicative-exponential linear logic (MELL)

$$\frac{}{\bot \vdash 0}\ \text{NOP} \qquad \frac{E \vdash P \quad F \vdash Q}{E \parr F \vdash P \mid Q}\ \text{PARA} \qquad \frac{E \leq F \quad F \vdash P}{E \vdash P}\ \text{SUB}$$

$$\frac{\vec{x} : A \otimes E \vdash P}{u : \downarrow A \otimes E \vdash u(\vec{x}).P}\ \text{IN} \qquad \frac{\vec{x} : A \otimes E^! \vdash P}{?u : \downarrow A \otimes E^! \vdash\ !u(\vec{x}).P}\ \text{IN!} \qquad \frac{E \vdash P}{u : \uparrow A \otimes (\vec{v} : A \parr E) \vdash \bar{u}\langle\vec{v}\rangle.P}\ \text{OUT}$$

$$\frac{(x : \uparrow A \parr x : \downarrow A) \otimes E \vdash P}{E \vdash (\boldsymbol{\nu} x : A^1)P}\ \text{NEW1} \qquad \frac{(!x : \uparrow A \parr ?x : \downarrow A) \otimes E \vdash P}{E \vdash (\boldsymbol{\nu} x : A^\omega)P}\ \text{NEW}\omega$$

- In the NEW rules, the name x must not occur in the environment E.
- In the IN rules, the names in $\vec{x}$ must not occur in the environments E and $E^!$.
- In IN!, $E^!$ stands for an environment of the shape $!y_1 : T_1 \otimes \ldots \otimes !y_n : T_n$.

Table 3: Typing rules for processes

Moreover, in applications of the SUB rule, we will keep the premiss $E \leq F$ implicit since E and F are the environments of the conclusion and premiss respectively.

4 **Lemma.** *A judgement $E \vdash P$ holds if and only if it has a derivation where* SUB *rules appear only right above* IN *and* NEW *rules and at the root of the proof.*

Proof. Firstly, it is clear that successive uses of the SUB rule can always be gathered into one thanks to the cut rule of MELL. We may thus assume without loss of generality that no SUB rule occurs above another SUB rule. Then one easily checks by case analysis on the proofs that each SUB rule can be commuted down with any rule except NEW and IN rules because these impose constraints on the context of their premiss. □

This lemma allows us to consider a restricted form of derivation when reasoning on typed processes. In order to establish subject reduction in the next section, we will also need the following general properties of MELL proofs:

5 **Lemma (substitutivity).** *If $\vdash \Gamma$ is a provable sequent in MELL, then for all propositional variable α and formula A the sequent $\vdash \Gamma[A/\alpha]$ is also provable.*

6 **Lemma (interpolation).** *Let Γ and Δ be two multisets of MELL formulas. If $\vdash \Gamma, \Delta$ is provable, then there exists a formula F that contains only literals present in both $\Gamma^\perp$ and Δ such that the sequents $\vdash \Gamma, F$ and $\vdash F^\perp, \Delta$ are provable.*

Both lemmas are easily proved by structural induction over proofs (a detailed proof for lemma 6 can be found in appendix A.1). They actually hold for full linear logic but we state them in MELL because it is the fragment we use in this paper.

2.2 Execution

Our presentation of execution uses structural congruence and reduction, because it provides simpler statements than a presentation using a labelled transition system.

7 **Definition (structural congruence).** The congruence $\equiv$ over process terms is defined by abelian monoid laws for parallel composition and the standard scoping rules:

$$(P \mid Q) \mid R \equiv P \mid (Q \mid R) \qquad P \mid Q \equiv Q \mid P \qquad P \mid 0 \equiv P$$
$$(\boldsymbol{\nu} x : A^k)(\boldsymbol{\nu} y : B^\ell)P \equiv (\boldsymbol{\nu} y : B^\ell)(\boldsymbol{\nu} x : A^k)P \qquad P \mid (\boldsymbol{\nu} x : A^k)Q \equiv (\boldsymbol{\nu} x : A^k)(P \mid Q)$$

where $x \neq y$ and x does not occur free in P in the last rule.

8 **Lemma.** *Typing is preserved by structural congruence.*

Proof. This is proved by checking each axiom of structural congruence. Most cases are direct, the only technical point is the proof that if $(\boldsymbol{\nu} x : A^k)(P \mid Q)$ is typeable and x does not occur in P, then $P \mid (\boldsymbol{\nu} x : A^k)Q$ has the same type. We transform a generic typing of $(\boldsymbol{\nu} x : A^k)(P \mid Q)$ into a typing of $P \mid (\boldsymbol{\nu} x : A^k)Q$ as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{E \vdash P \quad F \vdash Q}{E \parr F \vdash P \mid Q}\,\text{PARA}}{[x]^k_A \otimes G \vdash P \mid Q}\,\text{SUB}}{G \vdash (\boldsymbol{\nu} x : A^k)(P \mid Q)}\,\text{NEW}k}
\quad\rightarrow\quad
\dfrac{E \vdash P \qquad \dfrac{\dfrac{F \vdash Q}{[x]^k_A \otimes H \vdash Q}\,\text{SUB}}{H \vdash (\boldsymbol{\nu} x : A^k)Q}\,\text{NEW}k}{\dfrac{E \parr H \vdash P \mid (\boldsymbol{\nu} x : A^k)Q}{G \vdash P \mid (\boldsymbol{\nu} x : A^k)Q}\,\text{SUB}}\,\text{PARA}$$

In order to find the environment H, remark that the subtyping on the left is justified by an MELL proof of $\vdash ([x]^k_A)^\perp, G^\perp, E, F$. By lemma 6, there is a formula H such that $\vdash G^\perp, E, H$ and $\vdash H^\perp, ([x]^k_A)^\perp, F$ are provable and the literals in H occur both in $G, E^\perp$ and in $([x]^k_A)^\perp, F$. By hypothesis x does not occur in P hence not in E, and not in G either by the side-condition on NEWk, so there is no x in H. Hence the literals in H occur in F so they are positive and H is an environment type. The proofs of $\vdash G^\perp, E, H$ and $\vdash H^\perp, ([x]^k_A)^\perp, F$ justify the SUB rules on the right. The other cases are detailed in appendix A.2. □

9 **Definition (reduction).** Reduction is the relation $\xrightarrow{\ell}$ where ℓ is either a name or the symbol τ. It is generated by the rules

$$\bar{u}\langle\vec{v}\rangle.P \mid u(\vec{x}).Q \xrightarrow{u} P \mid Q[\vec{v}/\vec{x}] \qquad \bar{u}\langle\vec{v}\rangle.P \mid !u(\vec{x}).Q \xrightarrow{u} P \mid Q[\vec{v}/\vec{x}] \mid !u(\vec{x}).Q$$

extended to arbitrary contexts as

$$\frac{P \xrightarrow{\ell} P'}{P \mid Q \xrightarrow{\ell} P' \mid Q} \quad \frac{P \xrightarrow{\ell} P' \quad \ell \neq u}{(\boldsymbol{\nu} u)P \xrightarrow{\ell} (\boldsymbol{\nu} u)P'} \quad \frac{P \xrightarrow{u} P'}{(\boldsymbol{\nu} u : A^1)P \xrightarrow{\tau} P'} \quad \frac{P \xrightarrow{u} P'}{(\boldsymbol{\nu} u : A^\omega)P \xrightarrow{\tau} (\boldsymbol{\nu} u : A^\omega)P'}$$

and saturated under structural congruence.

The only difference with standard reduction is that we delete linear name creations as soon as their name is used. This is consistent with the linearity requirement, moreover in typed processes this requirement is fulfilled. In plain π-calculus this operation would be handled by the congruence rule $(\boldsymbol{\nu} x)P \equiv P$ if x is not free in P, but we choose not to use this approach here in order to avoid an extra kind of "new" operator just for this case.

10 **Theorem (subject reduction).** *For all typed term $\Gamma \vdash P$ and execution step $P \xrightarrow{\tau} P'$, the judgement $\Gamma \vdash P'$ is derivable.*

Proof. Thanks to lemma 8, we can reason up to structural congruence. For an interaction step on a linear channel, we have $(\boldsymbol{\nu} u : A^1)(\bar{u}\langle\vec{v}\rangle.P \mid u(\vec{x}).Q) \xrightarrow{\tau} P \mid Q[\vec{v}/\vec{x}]$. The left-hand side is typed as follows (using the simplification from lemma 4):

$$\dfrac{\dfrac{\dfrac{\dfrac{E \vdash P}{u : \uparrow A \otimes (\vec{v} : A \parr E) \vdash \bar{u}\langle\vec{v}\rangle.P}\,\text{OUT} \qquad \dfrac{\vec{x} : A \otimes F \vdash Q}{u : \downarrow A \otimes F \vdash u(\vec{x}).Q}\,\text{IN}}{(u : \uparrow A \otimes (\vec{v} : A \parr E)) \parr (u : \downarrow A \otimes F) \vdash \bar{u}\langle\vec{v}\rangle.P \mid u(\vec{x}).Q}\,\text{PARA}}{(u : \uparrow A \parr u : \downarrow A) \otimes H \vdash \bar{u}\langle\vec{v}\rangle.P \mid u(\vec{x}).Q}\,\text{SUB}}{H \vdash (\boldsymbol{\nu} u : A^1)(\bar{u}\langle\vec{v}\rangle.P \mid u(\vec{x}).Q)}\,\text{NEW1}$$

with the hypothesis that no name in $\vec{x}$ occurs in F. The SUB rule is justified by an MELL proof of $\vdash (((u : \uparrow A)^\perp \otimes (u : \downarrow A)^\perp) \parr H^\perp), (u : \uparrow A \otimes (\vec{v} : A \parr E)) \parr (u : \downarrow A \otimes F)$. By lemma 5, we can replace the atomic formula $u : \downarrow A$ by $\vec{v} : A$ and the atomic formula $u : \uparrow A$ by $(\vec{v} : A)^\perp$, then we get a proof of

$$\vdash (\vec{v} : A \otimes (\vec{v} : A)^\perp) \parr H^\perp, ((\vec{v} : A)^\perp \otimes (\vec{v} : A \parr E)) \parr (\vec{v} : A \otimes F)$$

The following sequents are easily proved in MELL:

$$\vdash H^\perp, (\vec{v} : A \parr (\vec{v} : A)^\perp) \otimes H$$
$$\vdash (\vec{v} : A \parr ((\vec{v} : A)^\perp \otimes E^\perp)) \otimes ((\vec{v} : A)^\perp \parr F^\perp), E \parr (\vec{v} : A \otimes F)$$

so we get a proof of $\vdash H^\perp, (\vec{v} : A \otimes F) \parr E$ by composition, which justifies the typing:

$$\dfrac{\dfrac{E \vdash P \qquad \vec{v} : A \otimes F \vdash Q[\vec{v}/\vec{x}]}{E \parr (\vec{v} : A \otimes F) \vdash P \mid Q[\vec{v}/\vec{x}]}\ \text{PARA}}{H \vdash P \mid Q[\vec{v}/\vec{x}]}\ \text{SUB}$$

The case of a reduction on a non-linear channel is similar, with some extra work to handle duplication; details can be found in appendix A.3. □

Remark that the introduction of negated atoms in the proof above makes us go through environment *formulas* that are not proper types, although composition by cut provides a subtyping between environment types. These intermediate steps correspond to the introduction in our system of axiom rules that transport arbitrary behaviours (here the $\vec{v} : A$) with no counterpart in the terms, as a decomposition of the name passing mechanism. This is similar to the central role of axioms in the proofs-as-schedules [3] paradigm.

2.3 The role of "new"

The subject reduction property is formulated for reductions on private channels, i.e. names that are explicitly created in the term. Indeed, the property fails without this assumption: not only is the type not preserved (which is expected in the case of linear capabilities), but communicated data may not have proper types. For instance, in a typed term like

$$\dfrac{\dfrac{E \vdash P}{u : \uparrow A \otimes (v : A \parr E) \vdash \bar{u}\langle v\rangle.P}\ \text{OUT} \qquad \dfrac{x : B \otimes F \vdash Q}{u : \downarrow B \otimes F \vdash u(x).Q}\ \text{IN}}{(u : \uparrow A \otimes (v : A \parr E)) \parr (u : \downarrow B \otimes F) \vdash \bar{u}\langle v\rangle.P \mid u(x).Q}\ \text{PARA}$$

the name v has type A but the name x has type B, and there is no reason that A and B are compatible, thus in general we cannot type the reduct $P \mid Q[v/x]$.

We do not consider this a serious defect of the system, it is mostly a matter of presentation. Indeed, the purpose of typing is to ensure proper composition of processes, and the creation of channels is part of the composition operation. Therefore, composition only makes sense in the presence of name creation operators, and in the example above neither NEW1 nor NEWω applies if A and B do not match. We could reformulate our system so that situations like the one above are forbidden by typing. A natural approach would be to enforce syntactic constraints on environment types, for instance that linear capability assignments occur at most once, that dual capability assignments have matching types, etc. We chose not to include such restrictions for simplicity, relying on the above justification.

2.4 Properties of typed processes

It can be proved that processes typed in our system are well-behaved:

11 **Theorem (termination).** *Typed processes have no infinite sequence of transitions on private names.*

12 **Theorem (lock-freeness).** *In every execution of a typed closed term, every active output eventually interacts with an input.*

Proofs of these facts can be obtained by realisability techniques, as in previous work by the author [1, 2], or by syntactic means by relating execution with the cut-elimination procedure of

linear logic. We do not include proofs because they are out of the scope of this paper. Nevertheless, a fundamental point in the arguments is that they rely on the consistency of linear logic (through the cut-elimination property). In relaxations of the system introduced in section 3, we will express systems which do not enjoy those properties, by means of inconsistent extensions of this logic.

2.5 Variations

The choice of the polyadic π-calculus in the presentation of our system is justified by the fact that it is very expressive and also very standard. However, we can adapt our approach to most variants of the calculus.

Asynchrony This is the restriction on outputs to have no continuations [7]. The typing of a free output atom $\bar{u}\langle\vec{v}\rangle$, considered as a simple process $\bar{u}\langle\vec{v}\rangle.0$, is as follows:

$$\dfrac{\dfrac{}{\bot \vdash 0}\ \text{NOP}}{u : \uparrow A \otimes (\vec{v} : A \parr \bot) \vdash \bar{u}\langle\vec{v}\rangle.0}\ \text{OUT} \quad \rightsquigarrow \quad \dfrac{}{u : \uparrow A \otimes \vec{v} : A \vdash \bar{u}\langle\vec{v}\rangle}\ \text{OUT-ASYNC}$$

where the simplified type is appropriate since it is linearly equivalent to the one on the left, because of neutrality of $\bot$ for $\parr$. Apart from this rule, nothing is changed in the system for the asynchronous π-calculus.

Internal mobility This is the restriction that output prefixes only communicate distinct bound names [26]. This simplifies the theory of the calculus and makes it symmetric like CCS. In our type system, we also get the symmetry in typing rules. For this purpose we can introduce duality over behaviour types:

13 **Definition (duality).** For a behaviour type A, the dual $\overline{A}$ is defined inductively as:

$$\begin{array}{llll} \overline{\uparrow A} := \downarrow A & \overline{!A} := ?\overline{A} & \overline{A \otimes B} := \overline{A} \parr \overline{B} & \overline{1} := \bot \\ \overline{\downarrow A} := \uparrow A & \overline{?A} := !\overline{A} & \overline{A \parr B} := \overline{A} \otimes \overline{B} & \overline{\bot} := 1 \end{array}$$

The dual $\overline{A}$ of a formula A is a form of linear negation, except that the dual of a capability $\uparrow A$ is the capability $\downarrow A$, whereas negations keep capabilities unaffected in our environment formulas. Note that we do not apply duality inside the capability, since we follow the approach of i/o types, where this convention is the appropriate one. Nevertheless, logically, the output capability contains a negation, as illustrated by the bound output rule below.

14 **Lemma (generalised new).** *The following rule is derivable, assuming the tuple $\vec{x}$ is made of pairwise distinct names:*

$$\dfrac{(\vec{x} : A \parr \vec{x} : \overline{A}) \otimes E \vdash P}{E \vdash (\boldsymbol{\nu}\vec{x})P}\ \text{NEW}^*$$

Proof. This is proved by induction A. The base case is when A is a linear or exponential capability, then one of the NEW rules applies directly. If $A = \bot$, then $\vec{x}$ is empty and we have $(\vec{x} : A \parr \vec{x} : \overline{A}) \otimes E = (\bot \parr 1) \otimes E \simeq 1 \otimes E \simeq E$ so the rule holds by linear equivalence. The case

$A = 1$ is similar. If $A = B \parr C$ then $\vec{x}$ splits as $\vec{y}, \vec{z}$ so that we have

$$\dfrac{\dfrac{\dfrac{((\vec{y} : B \parr \vec{z} : C) \parr (\vec{y} : \overline{B} \otimes \vec{z} : \overline{C})) \otimes E \vdash P}{(\vec{z} : C \parr \vec{z} : \overline{C}) \otimes (\vec{y} : B \parr \vec{y} : \overline{B}) \otimes E \vdash P}\ \text{SUB}}{(\vec{y} : B \parr \vec{y} : \overline{B}) \otimes E \vdash (\boldsymbol{\nu}\vec{z})P}\ \text{NEW}^*}{E \vdash (\boldsymbol{\nu}\vec{y})(\boldsymbol{\nu}\vec{z})P}\ \text{NEW}^*$$

where the SUB rule is justified by a simple MLL proof. The case $A = B \otimes C$ is similar. □

Using this lemma, we can derive a typing rule for bound output:

$$\dfrac{\dfrac{\dfrac{\vec{x} : \overline{A} \otimes E \vdash P}{u : \uparrow A \otimes (\vec{x} : A \parr (\vec{x} : \overline{A} \otimes E)) \vdash \bar{u}\langle\vec{x}\rangle.P}\ \text{OUT}}{u : \uparrow A \otimes (\vec{x} : A \parr \vec{x} : \overline{A}) \otimes E \vdash \bar{u}\langle\vec{x}\rangle.P}\ \text{SUB}}{u : \uparrow A \otimes E \vdash (\boldsymbol{\nu}\vec{x})\bar{u}\langle\vec{x}\rangle.P}\ \text{NEW}^* \quad \rightsquigarrow \quad \dfrac{\vec{x} : \overline{A} \otimes E \vdash P}{u : \uparrow A \otimes E \vdash \bar{u}(\vec{x}).P}\ \text{OUT-BOUND}$$

Fusions Our system can be extended to calculi with free input, such as the fusion calculus [24]. The appropriate formulation is with a preorder over names [16] generated by "arcs" a/b which are explicit substitution atoms. The logical meaning of an arc is an implication $!(a : T \multimap b : T)$ for any capability type T: it allows a capability on a to be used as a capability on b; the modality is because the substitution is permanently available. The typing rule would be an axiom like $?(a : T \otimes (b : T)^{\perp}) \vdash a/b$; this implies the handling of negative atoms, which may have an impact on the structure of the system. We defer the formal development of this extension to future work, since it exceeds the scope of the present paper.

3 Existing systems as fragments and extensions

In this section, we describe formally how our system can express known type systems for processes, using relaxations and identifying fragments. By *relaxation*, we mean that we add new logical rules to MELL in order to prove more subtypings. The resulting system need not be logically consistent, the minimal requirements are that the new rules preserve

- the interpolation lemma, so that typing is still preserved under structural congruence,
- the substitution lemma, so that subject reduction still holds.

3.1 Linearity and i/o types

We show here how our system can express plain i/o types à la Pierce and Sangiorgi [25] and their extension with linearity by Kobayashi, Pierce and Turner [22] (hereafter referred to as KPT). We develop the relationship only with KPT, since plain i/o types are its fragment without linear types. We refer the reader to the original paper for the notations.

15 **Definition.** Let $\mathscr{L}$ be the fragment of KPT where

- in channel types, only pure input or output capabilities are used,
- linear channel creations must create both capabilities,
- the boolean data type is not used.

The translation $[\![\cdot]\!]$ maps channel types of $\mathscr{L}$ to channel types, tuples of channel types of $\mathscr{L}$ to behaviour types and contexts of $\mathscr{L}$ to environment types as follows:

$$[\![!^1\vec{T}]\!] := \uparrow[\![\vec{T}]\!] \quad [\![?^1\vec{T}]\!] := \downarrow[\![\vec{T}]\!] \quad [\![!^\omega\vec{T}]\!] := !\uparrow[\![\vec{T}]\!] \quad [\![?^\omega\vec{T}]\!] := ?\downarrow[\![\vec{T}]\!]$$
$$[\![T_1 \dots T_n]\!] := [\![T_1]\!] \otimes \dots \otimes [\![T_n]\!]$$
$$[\![x : !^m\vec{T}]\!] := x : [\![!^m\vec{T}]\!] \quad [\![x : ?^m\vec{T}]\!] := x : [\![?^m\vec{T}]\!] \quad [\![x : \updownarrow^m\vec{T}]\!] := x : [\![!^m\vec{T}]\!] \parr x : [\![?^m\vec{T}]\!]$$
$$[\![x_1 : T_1, \dots x_n : T_n]\!] := [\![x_1 : T_1]\!] \otimes \dots \otimes [\![x_n : T_n]\!]$$

The restriction on channel types is of minor importance as it can be lifted by a simple coding: communicating channels with no capabilities is useless so it can be removed, and instead of communicating a channel with both capabilities, one can communicate each capability as distinct arguments. As for the restriction on channel creation, it is harmless since a channel created without both capabilities will never have any communication. The exclusion of booleans is simply because our system, for simplicity of presentation, does not include base data types; extension of the system with such types is not problematic.

16 **Theorem.** *A typing judgement $\Gamma \vdash P$ holds in $\mathscr{L}$ if and only if the judgement $[\![\Gamma]\!] \vdash P$ holds in our system extended with the logical equivalences*

$$A \otimes B \simeq A \parr B \qquad 1 \simeq \bot \qquad !A \simeq ?A$$

Sketch of proof. Using these equivalences, environment types, up to associativity, commutativity and neutrality, are just multisets of capability assignments of the shape $x : T$ or $!x : T$. Moreover, the multiplicity of each $!x : T$ does not matter. Similarly, behaviour types are now just tuples of capabilities. This provides a reverse mapping from our types to those of $\mathscr{L}$. Then it is easy to check that each typing rule in $\mathscr{L}$ can be derived in our system, which proves the direct implication. For the reverse implication, we just have to check that our rules are also valid in $\mathscr{L}$, only taking care of multiple occurrences of a name in an environment type by appropriate constraints on the use of contraction. □

The addition of the logical equivalences can be achieved by adding to the proof rules of MELL any rules that implement these equations as linear equivalences (as new axiom rules or as new introduction rules for the connectives involved; these methods are equivalent). It is not hard to check that this relaxation does preserve the interpolation and substitution lemmas. Of course, lock-freeness and termination are lost, and this is directly related to the fact that the equivalences make the logic inconsistent: cut elimination is lost.

3.2 Control, sequentiality, etc.

In a series of works [5, 6, 19, 28], Berger, Honda and Yoshida studied refinements of i/o types with linearity where various properties are enforced including sequentiality, strong normalisation, or the behaviour of functional computation with control. The latter system (hereafter called HYB, we refer the reader to the paper [19] for the notations) was put in precise correspondence with proof nets for polarised linear logic by Honda and Laurent [18] and this correspondence fits in our system.

17 **Definition.** The translation $[\![\cdot]\!]$ from HYB types to behaviour types, HYB contexts to environ-

ment types and processes to processes is defined as follows:

$$[\![(\vec{\tau})^?]\!] := \uparrow\overline{[\![\vec{\tau}]\!]}, \quad [\![(\vec{\tau})^!]\!] := \downarrow[\![\vec{\tau}]\!], \quad [\![\tau_1 \dots \tau_n]\!] := (![\![\tau_1]\!] \parr ?\overline{[\![\tau_1]\!]}) \otimes \dots \otimes (![\![\tau_n]\!] \parr ?\overline{[\![\tau_n]\!]})$$
$$[\![x : \tau_O]\!] := !x : [\![\tau_O]\!], \quad [\![x : \tau_I]\!] := !x : \overline{[\![\tau_I]\!]} \parr ?x : [\![\tau_I]\!],$$
$$[\![!x(y_1 \dots y_n).P]\!] := !x(y_1 y_1' \dots y_n y_n').[\![P]\!] \qquad \text{with } y_1', \dots, y_n' \text{ fresh},$$
$$[\![\bar{x}(y_1 \dots y_n)P]\!] := (\boldsymbol{\nu} y_1 \dots y_n)(\bar{x}\langle y_1 y_1 \dots y_n y_n\rangle \mid [\![P]\!]).$$

This translation is essentially the isomorphism between π-calculus types à la HYB and formulas of LLP, plus the capability indications. A crucial difference is that we have to code every communication of a single name as the communication of a pair for the input and the output capabilities, since in HYB an input type $(\vec{\tau})^!$ actually allows the presence of outputs, while our type system does not allow sending both capabilities as a single argument. Through this translation, we do capture HYB's typing, and the following theorem is proved by writing translations between the two systems:

18 **Theorem.** *A judgement* $\vdash P \triangleright x_1 : \tau_1, \dots, x_n : \tau_n$ *is derivable in HYB if and only if the judgement* $[\![x_1 : \tau_1]\!] \otimes \dots \otimes [\![x_n : \tau_n]\!] \vdash P$ *is derivable in our system extended with the equations* $A \otimes B \simeq A \parr B$ *and* $1 \simeq \bot$.

Again, the identification of dual connectives makes the underlying logic degenerate, and indeed the logic above does not ensure normalisation. Honda and Laurent enumerate several restrictions of this system: acyclicity of name dependence, input or output determinism, etc; in our system, these restrictions mean than we do not identify dual connectives, then the theorem above extends as an embedding of LLP/π^c into our system.

The same approach can be used to handle other type systems of the same family, we leave the formalisation of the correspondence for those systems to future work.

3.3 Session types

Caires and Pfenning [8] formulated an equivalence between dyadic session types [17] and intuitionistic linear logic, using a suitable interpretation of the connectives: $u : A \multimap B$ means "on u, receive a channel of type A then proceed according to B", dually $u : A \otimes B$ means "send a channel of type A, then proceed according to B". This implies that the type of a channel must change during an interaction, following the progress of the session. This seems to be incompatible with type systems in which a type is assigned to each channel in a static way, including the present work, however the same authors with DeYoung and Toninho [11] found a reformulation of their correspondence (hereafter called DCPT) in the asynchronous π-calculus where this contradiction vanishes. The trick is that these channels must never have more than one active occurrence per polarity and this can be turned into linearity by applying to synchronous processes a translation $[\![\cdot]\!]$ defined as follows:

$$[\![u(x).P]\!] := u(xu').[\![P]\!][u'/u] \qquad [\![\bar{u}\langle v\rangle.P]\!] := (\boldsymbol{\nu} u')(\bar{u}\langle vu'\rangle \mid [\![P]\!][u'/v])$$

where u' is a fresh name that represents the state of u at the next step of interaction. Of course, this translation does not make sense for general processes, but in the case of the interaction discipline enforced by session types, this transformation is perfectly adequate.

19 **Theorem.** *Let* $[\![\cdot]\!]$ *be the following translation from LL formulas to channel types:*

$$[\![1]\!] := \uparrow\bot \qquad [\![A \otimes B]\!] := \uparrow([\![A^\bot]\!] \parr [\![B^\bot]\!]) \qquad [\![!A]\!] := \uparrow ?[\![A]\!]$$
$$[\![\bot]\!] := \downarrow\bot \qquad [\![A \parr B]\!] := \downarrow([\![A]\!] \parr [\![B]\!]) \qquad [\![?A]\!] := \downarrow ?[\![A]\!]$$
$$[\![x_1 : A_1, \dots, x_n : A_n]\!] := x_1 : [\![A_1]\!] \otimes \dots \otimes x_n : [\![A_n]\!]$$

If $\Gamma \vdash P :: x : A$ *is derivable in DCPT then* $[\![\Gamma]\!] \otimes x : [\![A^\perp]\!] \vdash P$ *holds in our system.*
If $[\![\Gamma]\!] \otimes x : [\![A^\perp]\!] \vdash P$ *holds, then* $\Gamma \vdash P' :: x : A$ *is derivable in DCPT for some* $P' \equiv P$.

Proof. The direct implication simply consists in checking that each rule of DCPT translates in our system, which is straightforward. For the reverse implication, we establish a standardisation result for our type system (applied to the π-calculus with internal mobility) which essentially eliminates the SUB rule by cut elimination; we just have to check that all permutations involved are structural congruences. □

4 Discussion

More systems Our results are formulated in a π-calculus without choice using MELL as a subtyping logic. We chose to present this system since it illustrates the fundamental ideas of our approach, but it can be naturally extended to a type system for the π-calculus with choice, more liberal replication, genericity [6] etc using full linear logic, with additives and second-order quantification.

We also conjecture that it should be possible to embed systems of a different kind using modalities different from the ! and ? of linear logic. In particular, type systems that ensure termination by stratification of names [10] should correspond to using our basic system but replacing MELL with a form of light logic [15] where the operations on exponentials are constrained using stratification techniques that are (at least superficially) similar.

Synchrony, or lack thereof The lock-freeness property that the system ensures is important but it implies a serious defect of our system: it is very weak at dealing with prefixing. A witness of this fact can be seen in the following derivation:

$$\dfrac{\dfrac{\dfrac{y : B \otimes E \vdash P}{v : \downarrow B \otimes E \vdash v(y).P}\ \text{IN}}{x : A \otimes F \vdash v(y).P}\ \text{SUB}}{u(x) : \downarrow A \otimes F \vdash u(x).v(y).P}\ \text{IN}$$

Assuming that the names u, v, x, y are all distinct, it is easy to prove (by reasoning on the MELL proof of $\vdash (x : A)^\perp \parr F^\perp, v : \downarrow B \otimes E$) that F can actually be written $v : \downarrow B \otimes F'$ up to associativity and commutativity, and that subsequently the subtypings $x : A \otimes F' \leq E$ and hence $x : A \otimes y : B \otimes F' \leq y : B \otimes E$ hold. Therefore the term $v(y).u(x).P$ will also by typeable by the same type as above. Hence our types are preserved by the equivalence

$$u(x).v(y).P \simeq v(y).u(x).P$$

The same argument applies to output prefixes and commutation between inputs and outputs. A consequence of this observation is that any typed equivalence over processes must include the rule above, in other words our type system actually tells about a very asynchronous calculus (this is nearly the calculus of solos [23] with restrictions on scopes, except that prefixes can freely commute but not interact).

A deep reason for this state of things is that the discipline on names in process composition stems from proof composition in linear logic, which fundamentally works by enforcing acyclicity and connectedness in connections between proofs [9], in a *commutative* context. Indeed, the multiplicative connectives can be interpreted as follows:

- $E \otimes F \vdash P$ means that P is expected to behave well in an environment that provides some behaviour for E and some behaviour for F, and those are *independent*.
- $E \parr F \vdash P$ means that P is expected to behave well when these two behaviours are *correlated*, i.e. some events in E can be prefixed by events in F and vice-versa.

With only this kind of information, there is no hope to have a type system that would accept $a.b \mid \bar{a}.\bar{b}$ but would reject $a.b \mid \bar{b}.\bar{a}$. The only way out of this problem is either to extend the logic with non-commutative connectives, or to introduce other forms of dependencies, for instance through quantification.

Semantics This paper does not discuss semantic aspects of logic and processes, however these are fundamental motivations of our approach. We claim that the method of starting with a very constrained system and the relaxing it in a controlled way using logical axioms should be fruitful in this respect.

Realisability can be used to extract interpretations of formulas and terms from syntax itself, using orthogonality as a generic form of testing. It is efficient, in particular, for specifying operational properties of processes, among which termination and lock-freeness. Capabilities get interpreted by basic operational definitions while logic is interpreted as in phase semantics, which justifies our use of entailment as subtyping since, in such semantics, $E \vdash F$ does imply the inclusion of E into F. Besides, consistency of phase interpretation accepts some axioms (like the mix rule or arbitrary weakening) but not others, which justifies the effects of adding those axioms in our subtyping logic.

Another promising direction is the use of denotational semantics of proofs as a way to build semantics of processes. Evidence for this can be found, for instance, in the relational model of linear logic: it is a non-trivial model of proofs, yet it supports the identification of opposite types, as used in section 3.1 to rebuild i/o types. Using an appropriate interpretation for capability types, this should provide meaningful denotational models for i/o-typed processes. Besides, the flexibility of the relational model makes it suitable to interpret differential linear logic, in which it is possible to formulate encodings of processes of the calculus of solos [12]. Our approach thus provides new tools for the study of denotational models of processes. This could for instance extend a line of work of Varacca and Yoshida [27] interpreting the π-calculus in event structures using logical constructs.

References

[1] Emmanuel Beffara. *Logique, réalisabilité et concurrence*. PhD thesis, Université Paris 7, 2005.

[2] Emmanuel Beffara. A concurrent model for linear logic. In *21st International Conference on Mathematical Foundations of Programming Semantics (MFPS)*, volume 155 of *Electronic Notes in Theoretical Computer Science*, pages 147–168, 2006.

[3] Emmanuel Beffara. A proof-theoretic view on scheduling in concurrency. In Paolo Oliva, editor, *Classical Logic and Computation 2014*, volume 164 of *Electronic Proceedings in Theoretical Computer Science*, pages 78–92, Wien, July 2014.

[4] Gianluigi Bellin and Philip J. Scott. On the π-calculus and linear logic. *Theoretical Computer Science*, 135(1):11–65, May 1994.

[5] Martin Berger, Kohei Honda, and Nobuko Yoshida. Sequentiality and the π-calculus. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications*, volume 2044 of *Lecture Notes in Computer Science*, pages 29–45. Springer Berlin Heidelberg, 2001.

[6] Martin Berger, Kohei Honda, and Nobuko Yoshida. Genericity and the π-calculus. *Acta Informatica*, 42(2-3):83–141, November 2005.

[7] Gérard Boudol. Asynchrony and the π-calculus. Research report, INRIA, 1992.

[8] Luís Caires and Frank Pfenning. Session types as intuitionistic linear propositions. In Paul Gastin and François Laroussinie, editors, *Proceedings of the 21st International Conference on Concurrency Theory*, volume 6269 of *Lecture Notes in Computer Science*, pages 222–236, Paris, France, August 2010. Springer Berlin Heidelberg.

[9] Vincent Danos and Laurent Regnier. The structure of multiplicatives. *Archive for Mathematical Logic*, 28(3):181–203, October 1989.

[10] Yuxin Deng and Davide Sangiorgi. Ensuring termination by typability. In *Exploring New Frontiers of Theoretical Informatics, IFIP 18th World Computer Congress, TC1 3rd International Conference on Theoretical Computer Science (TCS2004)*, pages 619–632. Kluwer, 2004.

[11] Henry DeYoung, Luis Caires, Frank Pfenning, and Bernardo Toninho. Cut reduction in linear logic as asynchronous session-typed communication. In Patrick Cégielski and Arnaud Durand, editors, *Computer Science Logic (CSL'12) - 26th International Workshop/21st Annual Conference of the EACSL*, volume 16 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 228–242, Dagstuhl, Germany, 2012. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[12] Thomas Ehrhard and Olivier Laurent. Acyclic solos and differential interaction nets. *Logical Methods in Computer Science*, 6(3), September 2010.

[13] Yuxi Fu. A proof-theoretical approach to communication. In Pierpaolo Degano, Roberto Gorrieri, and Alberto Marchetti-Spaccamela, editors, *24th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1256 of *Lecture Notes in Computer Science*, pages 325–335. Springer Verlag, 1997.

[14] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50(1):1–101, 1987.

[15] Jean-Yves Girard. Light linear logic. *Information and Computation*, 143(2):175–204, 1998.

[16] Daniel Hirschkoff, Jean-Marie Madiot, and Davide Sangiorgi. Name-passing calculi: from fusions to preorders and types. In *Logic in Computer Science (LICS), 2013 28th Annual IEEE/ACM Symposium on*, pages 378–387, Los Alamitos, CA, USA, 2013. IEEE Computer Society.

[17] Kohei Honda. Types for dyadic interaction. In Eike Best, editor, *CONCUR'93*, volume 715 of *Lecture Notes in Computer Science*, pages 509–523. Springer Berlin Heidelberg, 1993.

[18] Kohei Honda and Olivier Laurent. An exact correspondence between a typed π-calculus and polarised proof-nets. *Theoretical Computer Science*, 411(22–24):2223–2238, 2010.

[19] Kohei Honda, Nobuko Yoshida, and Martin Berger. Control in the π-calculus. In *4th ACM-SIGPLAN Continuation Workshop*, 2004.

[20] Kohei Honda, Nobuko Yoshida, and Marco Carbone. Multiparty asynchronous session types. In *Proceedings of the 35th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '08, pages 273–284, New York, NY, USA, 2008. ACM.

[21] Naoki Kobayashi. A type system for lock-free processes. *Information and Computation*, 177(2):122–159, September 2002.

[22] Naoki Kobayashi, Benjamin C. Pierce, and David N. Turner. Linearity and the pi-calculus. *ACM Transactions on Programming Languages and Systems*, 21(5):914–947, September 1999.

[23] Cosimo Laneve and Björn Victor. Solos in concert. In Jiří Wiederman, Peter van Emde Boas, and Mogens Nielsen, editors, *26th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1644, pages 513–523. Springer Verlag, 1999.

[24] Joachim Parrow and Björn Victor. The fusion calculus: Expressiveness and symmetry in mobile processes. In *13th IEEE Symposium on Logic in Computer Science (LICS)*, pages 176–185, 1998.

[25] Benjamin C. Pierce and Davide Sangiorgi. Typing and subtyping for mobile processes. In *Proceedings of Eighth Annual IEEE Symposium on Logic in Computer Science, 1993. LICS '93*, pages 376–385, 1993.

[26] Davide Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *Theoretical Computer Science*, 167(1–2):235–274, 1996.

[27] Daniele Varacca and Nobuko Yoshida. Typed event structures and the π-calculus. *Electronic Notes in Theoretical Computer Science*, 158:373–397, 2006.

[28] Nobuko Yoshida, Martin Berger, and Kohei Honda. Strong normalisation in the π-calculus. In *16th Annual IEEE Symposium on Logic in Computer Science, 2001.*, pages 311–322, 2001.

A Technical appendix

A.1 Interpolation lemma (lemma 6)

Proof. We reason by induction on a cut-free proof π of $\vdash \Gamma, \Delta$.

- If π is an axiom rule, then three cases may occur:
  - either Γ and Δ are equal and are a single formula, then $F := \Delta$ works,
  - or $\Gamma = A^\perp, A$ for some A and Δ is empty, then $F := \bot$ works,
  - or Γ is empty and $\Delta = A^\perp, A$ for some A, then $F := 1$ works.
- If π is a 1 rule then either $\Gamma = \emptyset$ and $\Delta = 1$ or $\Gamma = 1$ and $\Delta = \emptyset$. In either case, $\Gamma^\perp, \Delta$ is a singleton $\{F\}$ where F provides the expected result.
- It π ends with a $\bot$ rule, it has the shape

$$\frac{\vdash \Gamma', \Delta'}{\vdash \Gamma', \Delta', \bot}\ \bot \qquad \text{with} \qquad \begin{array}{ll} \Gamma = \Gamma', & \Delta = \bot, \Delta' \quad \text{or} \\ \Gamma = \Gamma', \bot, & \Delta = \Delta' \end{array}$$

  We can apply the induction hypothesis on $\vdash \Gamma', \Delta'$, yielding proofs of $\vdash \Gamma', F$ and $\vdash F^\perp, \Delta'$, and conclude by adding a $\bot$ rule on the appropriate side.

- If π ends with a $\otimes$ rule, it has the shape

$$\frac{\vdash \Gamma_1, A_1, \Delta_1 \qquad \vdash \Gamma_2, A_2, \Delta_2}{\vdash \Gamma_1, \Gamma_2, A_1 \otimes A_2, \Delta_1, \Delta_2}\otimes \qquad \text{with} \qquad \begin{array}{l} \Gamma = \Gamma_1, \Gamma_2, \quad \Delta = A_1 \otimes A_2, \Delta_1, \Delta_2 \quad \text{or} \\ \Gamma = \Gamma_1, \Gamma_2, A_1 \otimes A_2, \quad \Delta = \Delta_1, \Delta_2 \end{array}$$

  In the first case, we proceed as follows using the induction hypothesis on Γ_i and Δ_i, A_i for each i:

$$\frac{\dfrac{}{\vdash \Gamma_1, F_1}\,\text{IH} \qquad \dfrac{}{\vdash \Gamma_2, F_2}\,\text{IH}}{\vdash \Gamma_1, \Gamma_2, F_1 \otimes F_2}\otimes \qquad \frac{\dfrac{}{\vdash F_1^\perp, A_1, \Delta_1}\,\text{IH} \qquad \dfrac{}{\vdash F_2^\perp, A_2, \Delta_2}\,\text{IH}}{\dfrac{\vdash F_1^\perp, F_2^\perp, A_1 \otimes A_2, \Delta_1, \Delta_2}{\vdash F_1^\perp \parr F_2^\perp, A_1 \otimes A_2, \Delta_1, \Delta_2}\parr}\otimes$$

  These proofs provide the expected conclusions, with $F := F_1 \otimes F_2$. As for the constraints on literals, the induction hypothesis gives us that the atoms in each F_i are present both in $\Gamma_i^\perp$ and in Δ_i, A_i, hence the atoms in F are present both in $\Gamma^\perp$ and in Δ. The second case is similar except that we get $F := F_1 \parr F_2$.

- If π ends with a $\parr$ rule, it has the shape

$$\frac{\vdash \Gamma', A, B, \Delta'}{\vdash \Gamma', A \parr B, \Delta'}\parr \qquad \text{with} \qquad \begin{array}{l} \Gamma = \Gamma', \quad \Delta = A \parr B, \Delta' \quad \text{or} \\ \Gamma = \Gamma', A \parr B, \quad \Delta = \Delta' \end{array}$$

  In the first case we get a formula F and proofs of $\vdash \Gamma', F$ and $\vdash F^\perp, A, B, \Delta'$ by induction hypothesis, and from the second one we immediately deduce a proof of $\vdash F^\perp, A \parr B, \Delta'$, so the same F is appropriate. The second case is similar. The constraint on atoms is immediately satisfied.

- If π ends with a dereliction, weakening or contraction rule, we get the expected formula immediately by induction hypothesis on the premiss.

- If π ends with a promotion, it has the shape

$$\frac{\vdash ?\Gamma', A, ?\Delta'}{\vdash ?\Gamma', !A, ?\Delta'}! \qquad \text{with} \qquad \begin{array}{l} \Gamma = ?\Gamma', \quad \Delta = !A, ?\Delta' \quad \text{or} \\ \Gamma = ?\Gamma', !A, \quad \Delta = ?\Delta' \end{array}$$

  In the first case we get a formula F and proofs of $\vdash ?\Gamma, F$ and $\vdash F^\perp, A, ?\Delta'$, then by dereliction and promotion we get $\vdash ?\Gamma, !F$ and $\vdash ?F^\perp, !A, ?\Delta'$ (promotion on A) so $!F$ is appropriate. In the second case, similarly, we get $?F$ as the intermediate formula. □

A.2 Typing and structural congruence (lemma 8)

Proof. Thanks to lemma 4, it is enough to consider typing derivations where SUB rules only occur right above NEW rules (since no structural congruence rule involves inputs).

For associativity, commutativity and neutrality in parallel composition, the associated properties for $\parr$ and $\perp$ are easily provable in multiplicative linear logic.

For scope extrusion, consider a typed term $P \mid (\boldsymbol{\nu} x : A^k)Q$ where x does not occur in P. The typing derivation has the following shape:

$$\frac{E \vdash P \qquad \dfrac{[x]_A^k \otimes F \vdash Q}{F \vdash (\boldsymbol{\nu} x : A^k)Q}\,\text{NEW}k}{E \parr F \vdash P \mid (\boldsymbol{\nu} x : A^k)Q}\,\text{PARA}$$

where x does not occur in E (since it does not occur in P) nor in F (by the side condition in NEWk). Then we can write the following derivation:

$$\dfrac{\dfrac{\dfrac{E \vdash P \qquad [x]^k_A \otimes F \vdash Q}{E \parr ([x]^k_A \otimes F) \vdash P \mid Q}\,\text{PARA}}{[x]^k_A \otimes (E \parr F) \vdash P \mid Q}\,\text{SUB}}{E \parr F \vdash (\boldsymbol{\nu} x : A^k)(P \mid Q)}\,\text{NEW}$$

where the subtyping is easily proved in MLL. For the reverse rule, the typing of a term $(\boldsymbol{\nu} x : A^k)(P \mid Q)$ has the following shape:

$$\dfrac{\dfrac{\dfrac{E \vdash P \qquad F \vdash Q}{E \parr F \vdash P \mid Q}\,\text{PARA}}{[x]^k_A \otimes G \vdash P \mid Q}\,\text{SUB}}{G \vdash (\boldsymbol{\nu} x : A^k)(P \mid Q)}\,\text{NEW}$$

The subtyping judgement is a proof in MELL of $\vdash ([x]^k_A)^\perp \parr G^\perp, E \parr F$, which is equivalent to $\vdash ([x]^k_A)^\perp, G^\perp, E, F$. By lemma 6, we can deduce that there exists a MELL formula H such that $\vdash G^\perp, E, H$ and $\vdash H^\perp, ([x]^k_A)^\perp, F$ are provable and the literals in H occur both in $G, E^\perp$ and in $([x]^k_A)^\perp, F$. By hypothesis x does not occur in P so it does not occur in E, by the side-condition on NEWk it does not occur in G either, therefore x does not occur in H. Therefore the literals in H occur in F, which proves that H only has positive literals, so it is an environment type. The proofs of $\vdash G^\perp, E, H$ and $\vdash H^\perp, ([x]^k_A)^\perp, F$ induce subtypings $G \leq E \parr H$ and $[x]^k_A \otimes H \leq F$ so we can conclude this case by the following typing:

$$\dfrac{\dfrac{E \vdash P \qquad \dfrac{\dfrac{F \vdash Q}{[x]^k_A \otimes H \vdash Q}\,\text{SUB}}{H \vdash (\boldsymbol{\nu} x : A^k)Q}\,\text{NEW}}{E \parr H \vdash P \mid (\boldsymbol{\nu} x : A^k)Q}\,\text{PARA}}{G \vdash P \mid (\boldsymbol{\nu} x : A^k)Q}\,\text{SUB}$$

For commutation of restrictions, a typed term $(\boldsymbol{\nu} x : A^k)(\boldsymbol{\nu} y : B^\ell)P$ must have a derivation of the following shape:

$$\dfrac{\dfrac{\dfrac{[y]^\ell_B \otimes E \vdash P}{E \vdash (\boldsymbol{\nu} y : B^\ell)P}\,\text{NEW}}{[x]^k_A \otimes F \vdash (\boldsymbol{\nu} y : B^\ell)P}\,\text{SUB}}{F \vdash (\boldsymbol{\nu} x : A^k)(\boldsymbol{\nu} y : B^\ell)P}\,\text{NEW}$$

From $[x]^k_A \otimes F \leq E$ we deduce $[x]^k_A \otimes [y]^\ell_B \otimes F \leq [y]^\ell_B \otimes E$, so we have the following typing:

$$\dfrac{\dfrac{\dfrac{[y]^\ell_B \otimes E \vdash P}{[x]^k_A \otimes [y]^\ell_B \otimes F \vdash P}\,\text{SUB}}{[y]^\ell_B \otimes F \vdash (\boldsymbol{\nu} x : A^k)P}\,\text{NEW}}{F \vdash (\boldsymbol{\nu} y : B^\ell)(\boldsymbol{\nu} x : A^k)P}\,\text{NEW}$$

which validates the case of commutation. □

A.3 Subject reduction (theorem 10)

Proof. Thanks to lemma 8, we can reason up to structural congruence. For an interaction step between linear actions, we have $(\boldsymbol{\nu} u : A^1)(\bar{u}\langle\vec{v}\rangle.P \mid u(\vec{x}).Q) \to P \mid Q[\vec{v}/\vec{x}]$. The left-hand side is typed as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{E \vdash P}{u : \uparrow A \otimes (\vec{v} : A \parr E) \vdash \bar{u}\langle\vec{v}\rangle.P}\ \text{OUT} \qquad \dfrac{\vec{x} : A \otimes F \vdash Q}{u : \downarrow A \otimes F \vdash u(\vec{x}).Q}\ \text{IN}}{(u : \uparrow A \otimes (\vec{v} : A \parr E)) \parr (u : \downarrow A \otimes F) \vdash \bar{u}\langle\vec{v}\rangle.P \mid u(\vec{x}).Q}\ \text{PARA}}{[u]^1_A \otimes H \vdash \bar{u}\langle\vec{v}\rangle.P \mid u(\vec{x}).Q}\ \text{SUB}}{H \vdash (\boldsymbol{\nu} u : A^1)(\bar{u}\langle\vec{v}\rangle.P \mid u(\vec{x}).Q)}\ \text{NEW1}$$

with the hypothesis that no name in $\vec{x}$ occurs in F. The natural typing for the reduct is obtained as follows:

$$\dfrac{E \vdash P \qquad \vec{v} : A \otimes F \vdash Q[\vec{v}/\vec{x}]}{E \parr (\vec{v} : A \otimes F) \vdash P \mid Q[\vec{v}/\vec{x}]}\ \text{PARA}$$

By lemma 5, in the subtyping $[u]^1_A \otimes H \leq (u : \uparrow A \otimes (\vec{v} : A \parr E)) \parr (u : \downarrow A \otimes F)$ we can replace the atomic formula $u : \downarrow A$ by $\vec{v} : A$ and the atomic formula $u : \uparrow A$ by $(\vec{v} : A)^\perp$, then we get a proof of

$$\vdash ((\vec{v} : A)^\perp \otimes \vec{v} : A) \parr H^\perp, ((\vec{v} : A)^\perp \otimes (\vec{v} : A \parr E)) \parr (\vec{v} : A \otimes F)$$

The sequents

$$\vdash H^\perp, (\vec{v} : A \parr (\vec{v} : A)^\perp) \otimes H$$
$$\vdash (\vec{v} : A \parr ((\vec{v} : A)^\perp \otimes E^\perp)) \otimes ((\vec{v} : A)^\perp \parr F^\perp), E \parr (\vec{v} : A \otimes F)$$

are easily provable in MLL so by the cut rule we get a proof of $\vdash H^\perp, (\vec{v} : A \otimes F) \parr E$ by which we can conclude with the typing of the reduct:

$$\dfrac{\dfrac{\dfrac{E \vdash P \qquad \vec{v} : A \otimes F \vdash Q[\vec{v}/\vec{x}]}{E \parr (\vec{v} : A \otimes F) \vdash P \mid Q[\vec{v}/\vec{x}]}\ \text{PARA}}{H \vdash P \mid Q[\vec{v}/\vec{x}]}\ \text{SUB}}$$

For an interaction step involving a replicated input, we have

$$(\boldsymbol{\nu} u : A^\omega)(\bar{u}\langle\vec{v}\rangle.P \mid !u(\vec{x}).Q \mid R) \to (\boldsymbol{\nu} u : A^\omega)(P \mid Q[\vec{v}/\vec{x}] \mid !u(\vec{x}).Q \mid R).$$

The left-hand side is typed as follows:

$$\dfrac{\dfrac{\dfrac{\dfrac{E \vdash P}{u : \uparrow A \otimes (\vec{v} : A \parr E) \vdash \bar{u}\langle\vec{v}\rangle.P}\ \text{OUT} \qquad \dfrac{\vec{x} : A \otimes F^! \vdash Q}{?u : \downarrow A \otimes F^! \vdash !u(\vec{x}).Q}\ \text{IN!} \qquad G \vdash R}{(u : \uparrow A \otimes (\vec{v} : A \parr E)) \parr (?u : \downarrow A \otimes F^!) \parr G \vdash \bar{u}\langle\vec{v}\rangle.P \mid !u(\vec{x}).Q \mid R}\ \text{PARA}}{[u]^\omega_A \otimes H \vdash \bar{u}\langle\vec{v}\rangle.P \mid !u(\vec{x}).Q \mid R}\ \text{SUB}}{H \vdash (\boldsymbol{\nu} u : A^\omega)(\bar{u}\langle\vec{v}\rangle.P \mid !u(\vec{x}).Q \mid R)}\ \text{NEW}\omega$$

Then we can deduce the following typing for the reduct without $(\boldsymbol{\nu} u : A^\omega)$:

$$\dfrac{E \vdash P \qquad \vec{v} : A \otimes F^! \vdash Q[\vec{v}/\vec{x}] \qquad \dfrac{\vec{x} : A \otimes F^! \vdash Q}{?u : \downarrow A \otimes F^! \vdash !u(\vec{x}).Q}\ \text{REP} \qquad G \vdash R}{E \parr (\vec{v} : A \otimes F^!) \parr (?u : \downarrow A \otimes F^!) \parr G \vdash P \mid Q[\vec{v}/\vec{x}] \mid !u(\vec{x}).Q \mid R}\ \text{PARA}$$

The instance of SUB in the first typing uses a proof of

$$\vdash (?(u : \uparrow A)^\perp \otimes !(u : \downarrow A)^\perp) \parr H^\perp, (u : \uparrow A \otimes (\vec{v} : A \parr E)) \parr (?u : \downarrow A \otimes F^!) \parr G$$

Let $O := u : \uparrow A$ (output), $I := u : \downarrow A$ (input) and $V := \vec{v} : A$ (value). The sequent above is

$$\vdash (?O^\perp \otimes !I^\perp) \parr H^\perp, (O \otimes (V \parr E)) \parr (?I \otimes F^!) \parr G$$

Because of the side condition in the rule NEWω the name u does not occur in H so there is no other occurrence of the literal $O^\perp$ in the above sequent, hence the linear atom O can only be introduced as follows, up to permutations of rules:

$$\dfrac{\dfrac{}{\vdash O^\perp, O}\,\text{AX}}{\vdash ?O^\perp, O}\,?$$

If we replace this with

$$\dfrac{\dfrac{}{\vdash V^\perp, V}\,\text{AX}}{\vdash ?O^\perp, V^\perp, V}\,\text{W}$$

and we introduce a $\parr$ between $?O^\perp$ and $V^\perp$ just before $?O^\perp$ is involved in a tensor rule, we replace O with V in the proof above and we get a proof of

$$\vdash ((?O^\perp \parr V^\perp) \otimes !I^\perp) \parr H^\perp, (V \otimes (V \parr E)) \parr (?I \otimes F^!) \parr G$$

The subformula $?I$ is necessarily introduced by a (possibly η-expanded) axiom rule that introduces $!I^\perp$, besides the latter only occurs once so $?I$ is only introduced once and thus is not involved in any contraction (except possibly with formulas introduced by weakening, but this case can be eliminated), so if we replace this axiom by an axiom on any formula and get another valid proof. Using the formula $V^\perp \parr ?I$ we get

$$\vdash ((?O^\perp \parr V^\perp) \otimes (V \otimes !I^\perp)) \parr H^\perp, (V \otimes (V \parr E)) \parr ((V^\perp \parr ?I) \otimes F^!) \parr G$$

Composing this with the following proofs:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash ?O^\perp, !O}\,\text{AX} \quad \dfrac{}{\vdash V, V^\perp}\,\text{AX}}{\vdash ?O^\perp, !O \otimes V, V^\perp}\,\otimes \quad \dfrac{}{\vdash !I^\perp, ?I}\,\text{AX}}{\vdash ?O^\perp \otimes !I^\perp, !O \otimes V, V^\perp, ?I}\,\otimes}{\vdash ?O^\perp \otimes !I^\perp, (!O \otimes V) \parr (V^\perp \parr ?I)}\,\parr \quad \dfrac{}{\vdash H^\perp, H}\,\text{AX}}{\vdash ?O^\perp \otimes !I^\perp, H^\perp, ((!O \otimes V) \parr (V^\perp \parr ?I)) \otimes H}\,\otimes}{\vdash (?O^\perp \otimes !I^\perp) \parr H^\perp, ((!O \otimes V) \parr (V^\perp \parr ?I)) \otimes H}\,\parr$$

and

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\vdash V^\perp, V}\,\text{AX} \quad \dfrac{}{\vdash (F^!)^\perp, F^!}\,\text{AX}}{\vdash V^\perp, (F^!)^\perp, V \otimes F^!}\,\otimes \quad \dfrac{\dfrac{}{\vdash !I^\perp, ?I}\,\text{AX} \quad \dfrac{}{\vdash (F^!)^\perp, F^!}\,\text{AX}}{\vdash !I^\perp, (F^!)^\perp, ?I \otimes F^!}\,\otimes}{\vdash V^\perp \otimes !I^\perp, (F^!)^\perp, (F^!)^\perp, V \otimes F^!, ?I \otimes F^!}\,\otimes}{\vdash V^\perp \otimes !I^\perp, (F^!)^\perp, V \otimes F^!, ?I \otimes F^!}\,\text{C}}{\vdash (V^\perp \otimes !I^\perp) \parr (F^!)^\perp, V \otimes F^!, ?I \otimes F^!}\,\parr \quad \dfrac{\dfrac{}{\vdash V^\perp, V \otimes E^\perp, E}\,\text{AX,AX,}\otimes}{\vdash V^\perp \parr (V \otimes E^\perp), E}\,\parr}{\vdash (V^\perp \parr (V \otimes E^\perp)) \otimes ((V^\perp \otimes !I^\perp) \parr (F^!)^\perp), E, V \otimes F^!, ?I \otimes F^!}\,\otimes \quad \dfrac{}{\vdash G^\perp, G}\,\text{AX}}{\vdash (V^\perp \parr (V \otimes E^\perp)) \otimes ((V^\perp \otimes !I^\perp) \parr (F^!)^\perp) \otimes G^\perp, E, V \otimes F^!, ?I \otimes F^!, G}\,\otimes}{\vdash (V^\perp \parr (V \otimes E^\perp)) \otimes ((V^\perp \otimes !I^\perp) \parr (F^!)^\perp) \otimes G^\perp, E \parr (V \otimes F^!) \parr (?I \otimes F^!) \parr G}\,\parr$$

we get

$$\vdash (?O^\perp \otimes !I^\perp) \parr H^\perp, E \parr (V \otimes F^!) \parr (?I \otimes F^!) \parr G$$

that is

$$\vdash (?(u : \uparrow A)^\perp \otimes !(u : \downarrow A)^\perp) \parr H^\perp, E \parr (\vec{v} : A \otimes F^!) \parr (?(u : \downarrow A) \otimes F^!) \parr G$$

hence we have

$$\dfrac{\dfrac{\dfrac{E \vdash P \quad \vec{v} : A \otimes F^! \vdash Q[\vec{v}/\vec{x}] \quad \dfrac{\vec{x} : A \otimes F^! \vdash Q}{?u : \downarrow A \otimes F^! \vdash !u(\vec{x}).Q}\,\text{REP} \quad G \vdash R}{E \parr (\vec{v} : A \otimes F^!) \parr (?u : \downarrow A \otimes F^!) \parr G \vdash P \mid Q[\vec{v}/\vec{x}] \mid !u(\vec{x}).Q \mid R}\,\text{PARA}}{(?(u : \uparrow A) \parr !(u : \downarrow A)) \otimes H \vdash P \mid Q[\vec{v}/\vec{x}] \mid !u(\vec{x}).Q \mid R}\,\text{SUB}}{H \vdash (\boldsymbol{\nu} u : A^\omega)(P \mid Q[\vec{v}/\vec{x}] \mid !u(\vec{x}).Q \mid R)}\,\text{NEW}\omega$$

which concludes the proof. □